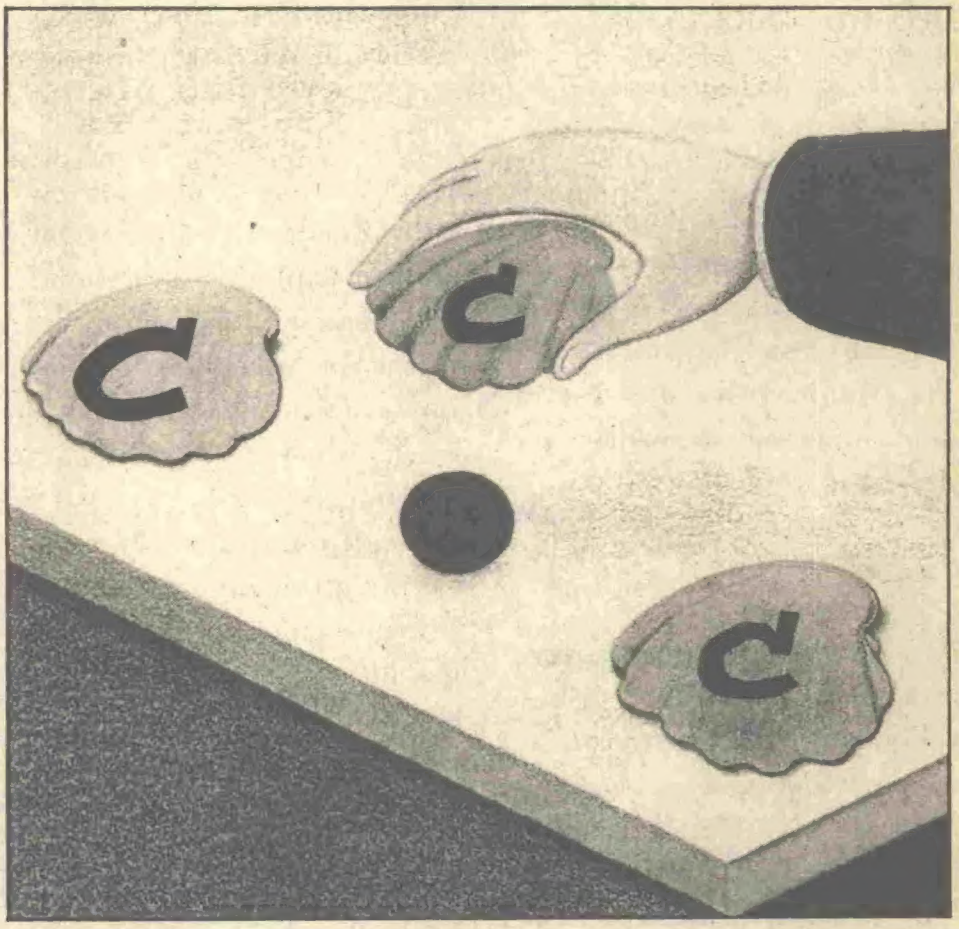
กิติพงษ์ ฤทธิบุตร

เรียน/เขียน TSR กับ C ตอนที่ 1

ศึกษาการเขียนโปรแกรมอีกรูปแบบหนึ่ง เพื่อใช้งานร่วมกันกับดอส โดยอาศัยภาษาซี



โปรแกรมจัดการระบบในเครื่องไมโครคอมพิวเตอร์ทั้งหลาย คงจะไม่มีโปรแกรมชุดไหนเป็นที่นิยมเท่ากับ DOS (Disk Operating System) ของไอบีเอ็ม ซึ่งถูกพัฒนาขีดความสามารถมาโดยตลอด จนปัจจุบันก็ได้พัฒนาขึ้นมาจนถึงเวอร์ชัน 5.0 แล้วความสำเร็จของดอสนั้นก็คงเป็นเพราะ สามารถตอบสนองความต้องการของผู้ใช้เครื่องคอมพิวเตอร์ได้เป็นอย่างดี โดยยึดหลักการของการให้มีการทำงานกับผู้ใช้คนเดียว และทำงานเพียงครั้งละหนึ่งโปรแกรม แต่ในเวอร์ชันใหม่ ๆ (ตั้งแต่ 4.0) ได้เปลี่ยนแนวมาทำงานในลักษณะหลาย ๆ งานพร้อมกัน (multitasking) ด้วย ตามความนิยมและความต้องการที่เปลี่ยนแปลงไป

อย่างไรก็ตามสิ่งที่ดอสขาดไปและไม่เคยได้ให้ความสนใจคือ เรื่องของการสนับสนุนการทำงานของโปรแกรมที่ฝังตัวอยู่ในหน่วยความจำ (RAM-resident) หรืออย่างที่เรียกกันโดยทั่วไปว่า TSRs เพราะโปรแกรมเหล่านี้จะใช้ฟังก์ชันของดอสคือ Terminate-and-Stay-Resident เพื่อจะให้สามารถฝังตัวอยู่ในหน่วยความจำได้และรอการเรียกใช้ต่อไป

โดยทั่วไปแล้วการเรียกใช้โปรแกรมประเภทนี้ (pop-up TSR) จะทำโดยผ่านแป้น (key) ที่กำหนดไว้เป็นกรณีพิเศษและเรียกมันว่า 'hot key' ซึ่งเมื่อถูกเรียกใช้มันก็จะทำการขัดจังหวะ (interrupt) การทำงานอื่น ๆ ที่กำลังทำอยู่ เพื่อทำงานตามหน้าที่ของมันจนเสร็จ และเมื่อออกจากการทำงานของโปรแกรมแล้ว ก็จะคืนการควบคุมให้กับโปรแกรมที่ทำการขัดจังหวะการทำงานไว้ เพื่อให้ทำงานต่อจากเดิมได้ตามปกติ

การที่ดอสไม่ให้ความสำคัญกับการใช้งานโปรแกรมประเภท TSR (โดยไม่มีการให้รายละเอียดเกี่ยวกับฟังก์ชันต่าง ๆ ของดอสเพื่อสนับสนุนการสร้างโปรแกรมเหล่านี้เอาไว้เลย) ทำให้การศึกษาค้นคว้าเพื่อให้สามารถใช้ TSR ได้นั้นจึงเป็นไปในลักษณะต่างคนต่างทำ อย่างไรก็ตามในช่วงเวลาที่ผ่านมา

คุณสมบัติข้อนี้ของดอสก็ได้ถูกค้นพบ ซึ่งทำให้สามารถสร้าง TSR ให้สามารถทำงานกับดอสได้อย่างถูกต้อง ปลอดภัย ถึงแม้จะไม่มีมาตรฐานเดียวกัน ในการเขียนโปรแกรมประเภทนี้

DOS กับ TSRs

เรื่องราวที่เราจะได้พูดถึงกันในคราวนี้ ก็จะเป็นเรื่องของคุณสมบัติดอสที่จะนำมาสร้างโปรแกรม TSR เกี่ยวกับวิธีการขั้นพื้นฐานที่จะทำให้สามารถสร้าง TSR ให้ทำงานได้อย่างถูกต้องและที่สำคัญคือ จะมาดูกันถึงวิธีการสร้างโปรแกรม TSR ที่สามารถใช้งานได้จริง ๆ โดยที่จะให้โปรแกรม TSR ที่จะสร้างเป็นตัวอย่างทำงานในลักษณะต่าง ๆ ดังนี้

* pop-up help system
* file list utility
* directory list utility
* number base converter

ในส่วนของ pop-up help system นั้นสามารถที่จะแสดง help screens (ให้คำแนะนำการใช้งานบนจอภาพ) ในหัวข้ออะไรก็ได้ตามที่ต้องการ นอกจากนี้ยังสามารถนำไปใช้งานร่วมกับโปรแกรม (application) อื่น ๆ โดยนำไปใช้ทำหน้าที่แสดงข้อความ (help information) ที่ตรงกับความต้องการสำหรับโปรแกรมนั้น ๆ ส่วนโปรแกรมอื่นที่จะทำหน้าที่ในส่วนที่เหลือ ก็จะแสดงให้เห็นถึงหลักการสำคัญ ๆ ที่ใช้สำหรับโปรแกรม TSR ทั้งหลาย ซึ่งได้แก่ การใช้ DOS และ BIOS (Basic Input-Output System) ใน TSR และการเรียกใช้ไฟล์ที่ต้องการจาก TSR

สำหรับโปรแกรมที่แสดงไว้ในส่วนต่าง ๆ ของเนื้อเรื่อง ก็จะเขียนตามกฎเกณฑ์ของคอมพายเลอร์เทอร์โบซี เพราะการเขียนโปรแกรมประเภท TSR นี้จำเป็นต้องใช้ขีดความสามารถของคอมพายเลอร์ที่มีอยู่เฉพาะในบางคอมพายเลอร์เท่านั้น (non-ANSI standard features) และในกรณีนี้จะใช้ของเทอร์โบซี นอกจากนี้การให้การสนับสนุนการเขียนโปรแกรม TSR นั้น จะมีไม่เท่ากันในคอมพายเลอร์แต่ละชุด ซึ่งคอมพายเลอร์ภาษา C ที่ให้การสนับสนุนการเขียนโปรแกรมในลักษณะนี้เป็นอย่างดี ฉะนั้นถ้ามีไมโครซอฟต์ซีก็สามารถที่จะเปลี่ยนโปรแกรมตัวอย่างนี้ให้สามารถคอมพายล์ด้วยไมโครซอฟต์ซีก็ได้ ซึ่งมีเพียงไม่กี่ฟังก์ชันเท่านั้นที่จะต้องแก้ไข (เปลี่ยนชื่อ) ส่วนข้อดีของเทอร์โบซีอีกอย่างหนึ่งก็คือ สามารถที่จะเขียนโปรแกรม TSR ได้โดยไม่ต้องใช้ภาษาแอสเซมบลีเลย

อย่างไรก็ตามการที่จะทำให้สามารถติดตามเนื้อหาของบทความได้อย่างดี ก็ควรจะมีความเข้าใจเกี่ยวกับ DOS และ BIOS บ้างพอควร ฉะนั้นถ้ายังไม่เข้าใจเกี่ยวกับเรื่องเหล่านี้มากนักก็ไม่ควรจะเปลี่ยนแปลง แก้ไขโปรแกรม (หรือถ้าทำก็ควรทำด้วยความระมัดระวัง) อีกประการหนึ่งคือจะมีเนื้อหาทางเทคนิคเกี่ยวกับการเขียนโปรแกรม TSR อยู่ในหลาย ๆ ส่วน ซึ่งอาจจะทำความเข้าใจได้ยาก แต่ก็สามารถจะใช้เป็นแนวทางในการสร้างโปรแกรม TSR ขึ้นเองได้

ความยุ่งยากในการเขียน TSRs

สิ่งที่ทำให้การเขียนโปรแกรม TSRs มีความยุ่งยากกว่าการเขียนโปรแกรมในแบบธรรมดาก็คือ ดอสไม่ยอมให้มีการเรียกใช้ฟังก์ชันเดิมซ้ำถ้ามีการใช้งานอยู่แล้ว (reentrant) ซึ่งหมายความว่าในขณะที่ดอสกำลังทำงานให้กับโปรแกรมใดโปรแกรมหนึ่งอยู่มันจะไม่ยอมให้มีการแทรกการทำงานจากโปรแกรมอื่น ถึงแม้ดอสจะไม่ยอมให้มีการเรียกใช้ซ้ำ เราก็ยังสามารถที่จะใช้วิธีการกำหนดค่า flag ให้เมื่ออยู่ในสถานการณ์ที่ปลอดภัยพอที่จะขัดจังหวะการทำงานได้ โดยจะต้องคืนค่า flag ให้เป็นอย่างเดิม เมื่อการทำงานไม่อยู่ในสภาวะที่จะขัดจังหวะได้ และวิธีการที่ TSR จะตรวจสอบว่าเป็นสถานการณ์ที่สามารถขัดจังหวะการทำงานของดอสได้อย่างปลอดภัยหรือไม่ ก็ให้ตรวจสอบที่ flag และค่าอินเตอร์รัพต์

ปัญหาอีกอย่างหนึ่งระหว่างดอสและ TSRs คือส่วนที่เกี่ยวข้องกับการทำงานกับดิสก์ เพราะดอสจะส่งข้อมูลที่อยู่ใน DTA (Disk Transfer Area) ให้กับแต่ละโปรแกรม ซึ่ง DTA นี้ก็จะเป็นพื้นที่ส่วนหนึ่งของ PSP (Program Segment Prefix) ที่จะมีอยู่ในทุกโปรแกรม และในกรณีที่ TSR ขัดจังหวะการทำงานของโปรแกรมอื่น มันก็จะเข้าไปใช้งาน PSP ของโปรแกรมนั้นด้วย ฉะนั้นถ้า TSR จะทำงานที่เกี่ยวข้องกับดิสก์ก็จะทำให้เสี่ยงต่อการเขียนทับข้อมูลของโปรแกรมนั้นที่อยู่ใน DTA ด้วย จึงทำให้มีความจำเป็นที่โปรแกรม TSR แต่ละชุดจะต้องมี DTA ของตัวเอง เพื่อใช้ในกรณีที่ต้องทำงานกับดิสก์ แล้วจึงกลับไปใช้งาน DTA ของโปรแกรมเดิมเมื่อทำงานเสร็จแล้ว

นอกจากนี้ก็ยังมีปัญหาเกี่ยวกับการใช้งาน TSR ที่อาจเกิดขึ้นได้เมื่อมี TSR ตั้งแต่สองชุดอยู่ในหน่วยความจำพร้อมกัน ถึงแม้ว่า TRS แต่ละชุดนั้นสามารถทำงานได้อย่างถูกต้องภายใต้ดอส และสิ่งที่มักจะก่อให้เกิดปัญหา

ในกรณีนี้ก็คือ 'hot key' (แป้นตัวอักษรที่จะใช้เรียก TSR) ซึ่งมักจะมีการกำหนดไว้ซ้ำกัน

สิ่งสุดท้ายที่เป็นปัญหากับ TSRs คือในขณะที่โปรแกรมกำลังใช้งานอยู่ก่อนหน้าการเรียกใช้ TSR นั้น ใช้วิธีการสั่งงาน (ทำงาน) โดยไม่ผ่าน DOS (หรือแม้แต่บางครั้งก็ไม่ผ่าน BIOS ด้วย) เช่นอาจทำการติดต่อกับดิสก์โดยไม่ผ่าน DOS (DOS high-level functions) และ BIOS (BIOS low-level functions) ซึ่งในกรณีนี้ถ้ามีการเรียกใช้ TSR เพื่อให้ติดต่อกับดิสก์อาจจะทำให้เกิดความเสียหายขึ้นได้ และนี่ก็เป็นเหตุผลสำคัญที่ทำให้ TSR ไม่สามารถจะตรวจสอบได้อย่างถูกต้องว่าสถานการณ์ที่เป็นอยู่ในขณะนั้น ๆ มีความปลอดภัย 100% อย่างไรก็ตามโปรแกรม TSR ที่จะได้พูดถึงต่อไปนี้ก็สามารถที่จะทำงานได้ด้วยความปลอดภัยมากที่สุด

ประตูใหญ่ของ TSR

โดยปกติแล้วการเรียกใช้ TSR นั้นจะต้องทำผ่าน 'interrupt' ซึ่งจะทำได้ด้วยการใช้แป้นพิมพ์ (keystroks interrupt) ฉะนั้นสิ่งสำคัญที่จะต้องรู้ในการเขียนโปรแกรม TSR คือ เรียนรู้การทำงานของ interrupt ต่าง ๆ ของโปรเซสเซอร์ ซึ่งในที่นี้ก็คือ โปรเซสเซอร์ในตระกูล 8086 ทั้งหลาย (รวมถึง 8088, 80186, 80286, 80386, และ 80486 ด้วย)

สำหรับโปรเซสเซอร์ 8086 นี้จะมีอินเตอร์รัพต์อยู่ทั้งหมด 256 ชุด ซึ่งเรียกว่า 'vectored interrupt' ที่จะทำหน้าที่เรียกใช้ 'Interrupt Service Routine (ISR)' โดยใช้ตำแหน่งที่เก็บชุดคำสั่งเอาไว้ และ 8086 จะใช้ 1024 (1K) ไบต์แรกเก็บตำแหน่งของ ISRs ซึ่งพื้นที่ส่วนนี้มักจะถูกเรียกเป็น 'interrupt vector table'

ในแต่ละตำแหน่งของอินเตอร์รัพต์ก็คือค่าเซกเมนต์และออฟเซตซึ่งจะต้องใช้เนื้อที่จำนวน 4 ไบต์ ฉะนั้นตำแหน่งของ ISR ของอินเตอร์รัพต์ 0 ก็คือ '0000:0000' (segment : offset) และอินเตอร์รัพต์ 1 ก็จะอยู่ที่ตำแหน่ง 0000:0004 เป็นอย่างนี้เรื่อย ๆ ไป ตัวอย่างเช่น สมมติให้ตำแหน่งของ ISR ที่อินเตอร์รัพต์ 2 ใช้อยู่คือ 1000:0800 ก็จะถูกเก็บเอาไว้ใน interrupt vector table โดยเริ่มต้นที่ตำแหน่ง '0000:0008' และทุกครั้งที่มีการเรียกใช้อินเตอร์รัพต์นี้ก็จะเรียกใช้คำสั่ง (rountine) ที่อยู่ในตำแหน่ง '1000:0800'

ในอินเตอร์รัพต์จำนวน 256 ชุดที่มีให้ใช้ได้นี้ จะมีอยู่บางชุดที่ดอสไม่ได้นำเอาไปใช้งานเช่นอินเตอร์รัพต์ 0×64 ซึ่งจะไม่มีการนำเอาไปใช้ในดอสทุกเวอร์ชันจนถึงเวอร์ชัน 4.0 เพราะฉะนั้นจึงสามารถที่จะนำเอาฟังก์ชันนี้มาใช้ประโยชน์ได้ ซึ่งการเขียนโปรแกรม TSR ในบทความนี้ก็จะนำเอาอินเตอร์รัพต์ 0×64 มาใช้เป็น flag เพื่อแสดงให้รู้ว่า TSR ถูกเรียกใช้ (load) แล้วหรือยัง

สำหรับ interrupt service routine ทั้งหมดจะต้องเรียกใช้ในแบบ 'FAR' เพราะโดยปกติแล้วมันจะเลิกการทำงานด้วยคำสั่ง 'return from interrupt' (IRET) นอกจากนี้ ISR ต่าง ๆ จะต้องเก็บค่าของรีจีสเตอร์ทั้งหมดเอาไว้ด้วย ซึ่งหมายความว่าเมื่อเริ่มต้นทำงาน TSR จะต้องเก็บ (PUSH) ค่าของรีจีสเตอร์แต่ละตัวเอาไว้ในสแต็ก และเมื่อออก

จากการทำงานแล้ว ก็จะต้องนำเอาค่าเหล่านั้นออกมา (POP) ไว้อย่างเดิม

การทำงานของ TSR นั้นจำเป็นต้องใช้อินเตอร์รัพต์เป็นจำนวนอย่างน้อย 1 ชุด เพื่อใช้สำหรับ interrupt service routines ของตัวเอง ซึ่งเราจะได้พูดถึงวิธีการเหล่านี้ต่อไป

อินเตอร์รัพต์กับเทอร์โบซี

คอมพายเลอร์เทอร์โบซี (รวมทั้งคอมพายเลอร์ชุดอื่น ๆ บางชุดที่ใช้งานบนดอส) ได้สนับสนุนการเขียนโปรแกรมประเภท TSR โดยได้สร้างชนิดของฟังก์ชัน (type modifier) เอาไว้ให้ใช้สำหรับงานนี้โดยเฉพาะคือ 'interrupt' และเมื่อมีการกำหนด (declare) ให้ฟังก์ชันไหนเป็น 'interrupt' ก็จะเป็นการบอกให้คอมพายเลอร์จัดเตรียมรีจีสเตอร์ไว้ใช้งาน และให้ออกจากการทำงานของฟังก์ชัน (terminats) ด้วยคำสั่ง 'IRET'

สำหรับเทอร์โบซีนั้นฟังก์ชันที่ถูกกำหนดให้เป็น 'interrupt' จะส่งผ่านรีจิสเตอร์ทั้งหมดในลักษณะของพารามิเตอร์ อย่างไรก็ตามผู้เขียนโปรแกรมสามารถที่จะกำหนดค่าของฟังก์ชันแบบนี้ให้เป็น 'void' ได้ (ไม่ต้องใช้ค่าพารามิเตอร์) ถ้าไม่ต้องการจะใช้ค่าต่าง ๆ ที่อยู่ในรีจิสเตอร์ ซึ่งในโปรแกรมที่เราจะได้เห็นกันต่อไปนั้นก็จะมีการใช้งานในลักษณะนี้อยู่

แต่ถ้าคอมพายเลอร์ภาษาซีที่ใช้อยู่ไม่มีรูปแบบการกำหนดฟังก์ชันในลักษณะนี้ ก็จะต้องเขียนโปรแกรมโดยใช้ภาษาแอสเซมบลี หรืออย่างน้อยก็ต้องใช้ assembly language interface เพื่อจัดการกับโปรแกรม TSR นั้น

ทำความรู้จักกับ PSP กันก่อน

สิ่งสำคัญอีกอย่างหนึ่งที่มีความจำเป็นกับการเขียนโปรแกรม TSR ก็คือ Program Segment Prefix หรือที่มักเรียกกันว่า 'PSP' ซึ่งในโปรแกรม TSR ทั้งหมดที่เราจะพูดถึงกันต่อไปนั้นก็จะใช้ PSP ด้วย เพราะ PSP จะถูกนำเอาไปใช้เป็นส่วนที่ทำหน้าที่เป็น Disk Transfer Area (DTA) ซึ่งก็จะได้แก่ส่วนที่อยู่ในตำแหน่ง 0×80 ไบต์จากตำแหน่งเริ่มต้นของ PSP อีกอย่างหนึ่งก็คือมันจะทำหน้าที่เก็บข้อมูลต่าง ๆ ของโปรแกรม โดยจะใช้ตำแหน่ง 0×2C ไบต์ จากตำแหน่งเริ่มต้นของ PSP

นอกจากนี้ PSP ยังมีความสำคัญต่อ TSR ในส่วนของข้อมูลอื่น ๆ อีกหลายอย่างที่จำเป็นต้องใช้ (program-specific) และด้วยเหตุนี้จึงทำให้โปรแกรม TSR จำเป็นต้องตั้งค่าของ PSP ใหม่เมื่อถูกเรียกใช้ โดยจะคืนค่าเหล่านี้ให้ไว้อย่างเดิมเมื่อมันทำงานเสร็จแล้ว

เพื่อให้การเขียนโปรแกรม TSR ทำได้สะดวกขึ้น เทอร์โบซี (รวมทั้งคอมพายเลอร์อื่นอีกบางชุด) จึงได้ให้ตัวแปร (built-in variable) สำหรับใช้งานเพิ่มขึ้นอีกหนึ่งตัวคือ '__psp' ซึ่งจะทำหน้าที่เป็นตัวชี้ (far pointer) ไปที่ตำแหน่งเริ่มต้นของ PSP ของโปรแกรม (แล้วเราจะได้เห็นวิธีการใช้งานในช่วงต่อไป)

การออกแบบขั้นพื้นฐาน

ในการเขียนโปรแกรม TSR นั้นอาจแบ่งโปรแกรมออกเป็น 3 ส่วนใหญ่ ๆ คือส่วนเริ่มต้น (initialization part), ISR (Interrupt Service Routine) ซึ่งจะใช้เพื่อสนับสนุนการทำงานของ TSR และส่วนสุดท้ายคือส่วนของชุดคำสั่งต่าง ๆ ของโปรแกรม TSR เอง

ส่วนเริ่มต้นของ TSR : เมื่อมีการประมวลผลโปรแกรม TSR เป็นครั้งแรกนั้น มันจะทำงานหลัก ๆ 3 อย่างคือ

* กำหนดค่าเริ่มต้นให้กับค่าต่าง ๆ ของโปรแกรม เช่น กำหนดค่าเริ่มต้นให้กับ TSR เพื่อให้ทำการเรียกใช้ help file ที่จะเรียกขึ้นมาดูได้ด้วยวิธีการแบบ pop-up help system ซึ่งการทำงานของ TSR ในลักษณะนี้จะต้องทำงานได้เร็ว จึงจำเป็นต้องนำเอาไฟล์ที่จะต้องใช้งานเข้ามาเก็บไว้ในหน่วยความจำตลอดเวลา
* หน้าที่อีกอย่างหนึ่งของการทำงานในส่วนนี้ คือการจัด interrupt address ต่าง ๆ ใหม่ โดยให้ชี้ไปที่รูทีนต่าง ๆ ของ TSR
* ส่วนสุดท้ายคือการเลิกการทำงานด้วยการเรียกใช้ฟังก์ชัน 0×31 ของดอสเพื่อเป็นการออกจากการทำงานด้วยการฝังตัวอยู่ในหน่วยความจำ ซึ่งมีคอมพายเลอร์ภาษาซีอยู่หลายชุดที่มีฟังก์ชันของตัวเองเพื่อทำหน้าที่นี้ ส่วนเทอร์โบซีได้กำหนดฟังก์ชัน 'keep ( )' ไว้ให้ใช้เพื่อวัตถุประสงค์นี้ (ส่วนของไมโครซอฟต์จะใช้ฟังก์ชัน '__dos__keep ( )' ซึ่งคือการเรียกใช้ฟังก์ชัน 0×31 ของดอสนั่นเอง โดยมีรูปแบบการใช้งานดังนี้

  void keep: ซึ่งค่าของ 'status' นี้จะถูกส่งคืนให้กับดอสและจำนวนหน่วยความจำ (RAM) ที่จะจัดสรรให้กับ TSR นั้น จะกำหนดเอาไว้ในตัวแปรชื่อ 'mem__size' โดยกำหนดเป็นจำนวนพารากราฟ ซึ่งมีขนาด 16 ไบต์ต่อพารากราฟ ในกรณีคอมพายเลอร์ที่ใช้ไม่มีฟังก์ชันที่ทำหน้าที่ในลักษณะเช่นเดียวกับ 'keep( )' ก็จะต้องใช้ฟังก์ชัน 0×31 ของดอสด้วยการคืนค่าที่ต้องการในรีจิสเตอร์ 'AL' และขนาดของโปรแกรมจะใช้รีจิสเตอร์ 'DX' ซึ่งคอมพายเลอร์ส่วนใหญ่จะมีวิธีการเรียกใช้ฟังก์ชันของดอสอยู่แล้ว

ในการพิจารณาว่าจะใช้หน่วยความจำของแรมเป็นจำนวนเท่าใดสำหรับโปรแกรม TSR นั้น โดยปกติในคอมพายเลอร์จะมีคำสั่งที่ทำหน้าที่นี้อยู่แล้ว อย่างไรก็ตามจำนวนหน่วยความจำที่โปรแกรมต้องการใช้นั้นจะมีขนาดใหญ่กว่าขนาดของไฟล์ (.EXE') ฉะนั้นก็อาจจะกำหนดขนาดของหน่วยความจำที่ต้องการเอาไว้เป็นจำนวน 64 กิโลไบต์เลยก็ได้ ซึ่งตัวเลขนี้จะสามารถใช้ได้กับ TSR ที่สร้างขึ้นด้วยคอมพายเลอร์ตัวอื่น ๆ ด้วย

ถ้าในคอมพายเลอร์ที่ใช้ไม่มีคำสั่งที่จะใช้คำนวณจำนวนหน่วยความจำที่ TSR ต้องการใช้ก็ให้ใช้ผลรวมของข้อมูล (global data) ทั้งหมดกับขนาดของไฟล์ ('EXE' file) แล้วเผื่อเอาไว้อีกประมาณ 2,000-3,000 ไบต์เพื่อความปลอดภัย และถ้าพบว่า TSR ยังไม่สามารถทำงานได้อย่างสมบูรณ์ ก็ให้เพิ่มขนาดของหน่วยความจำให้อีก

จัดการกับ TSR: ส่วนที่สองของ TSR คือส่วนที่ใช้จัดการกับ TSR เพราะโปรแกรม TSR ส่วนใหญ่จะเรียกใช้ (activated) ได้จากการกด 'hot key' (แป้นพิมพ์ที่กำหนดไว้สำหรับเรียกใช้ TSR) ซึ่งวิธีการตรวจจับข้อมูลของแป้นพิมพ์นั้น จะทำได้ด้วยการแทนที่ key board input interrupt ด้วยรูทีนของมันเอง (ในส่วนที่ใช้ทำหน้าที่นี้) และโดยที่ TSR จะต้องตรวจจับ hot key ในระดับต่ำสุด เพราะฉะนั้นจึงต้องแทนที่ BIOS key board interrupt (interrupt 9) ด้วย key board ISR ของมันเอง และเก็บตำแหน่งของ interrupt 9 ISR เอาไว้โดยไม่เรียกใช้

เมื่อมีการกดแป้นพิมพ์แต่ละครั้งก็จะมีการเรียกใช้ interrupt 9 ซึ่งจะมีผลให้มีการเรียกใช้ key board ISR ที่จัดเตรียมไว้ และสิ่งแรกที่ TSR จะต้องทำคือการเรียกใช้บริการที่ทำหน้าที่เรียกข้อมูลที่ป้อนจากแป้นพิมพ์ ต่อจากนั้นก็จะต้องพิจารณาว่าเป็น 'hot key หรือไม่ ถ้าใช่ก็จะเรียกใช้ (pop-up) TSR ขึ้นมาทำงาน (เมื่อพบว่าอยู่ในสถานการณ์ที่ปลอดภัย) แล้วก็จะลบ hot key นั้นออกจากส่วนรับข้อมูลเพื่อไม่ให้ถูกนำมาใช้งานอีก

สิ่งที่จะต้องเพิ่มให้กับคีย์บอร์ดอินเตอร์รัพต์อีกอย่างหนึ่งคือ การเพิ่ม ISR อย่างน้อยอีก 1 ชุดให้กับ TSR เพื่อทำหน้าที่ตรวจสอบว่าอยู่ในสภาวะที่ปลอดภัยสำหรับการขัดจังหวะการทำงานของดอส ซึ่ง TSR ส่วนใหญ่จำเป็นต้องเข้าไปจัดการกับ timer interrupt (interrupt 8) เพื่อช่วยในการเรียกใช้ TSR ให้เป็นไปอย่างเหมาะสม นอกจากนี้ในบางสถานการณ์ก็อาจจำเป็นต้องตรวจสอบการทำงานของอินเตอร์รัพต์ตัวอื่น ๆ ด้วย

ทำงานตามหน้าที่ : เมื่ออยู่ในสถานการณ์ที่ปลอดภัยพอที่จะขัดจังหวะการทำงานของดอสได้ก็สามารถที่จะเรียกใช้ (pop-up) โปรแกรม TSR ได้ แต่อย่างไรก็ตามการประมวลผล TSR ก็จะไม่เหมือนกับโปรแกรมอื่น (normal program) ด้วยเหตุผลหลายอย่างด้วยกันคือ

* การที่ TSR สามารถจะถูกเรียกใช้ในระหว่างการทำงานของโปรแกรมอื่น จึงจำเป็นต้องมีวิธีการที่จะใช้เก็บสิ่งต่าง ๆ ที่อยู่บนจอภาพก่อนการเรียกใช้ TSR และจะต้องนำสิ่งที่เก็บไว้นั้นคืนให้กับจอภาพหลังจากที่ได้ทำงานเสร็จแล้ว
* โดยปกติแล้ว TSR จะไม่สามารถใช้คำสั่งที่ใช้วิธีการการจัดสรรเนื้อที่แบบไม่กำหนดจำนวนที่แน่นอนได้เช่น 'malloc( )' ในภาษาซี ซึ่งก็จะรวมไปถึงฟังก์ชันอื่น ๆ ที่เรียกใช้ฟังก์ชันนี้ (malloc( )') ในระหว่างทำงานด้วยเช่น 'printf( )' และ 'fopen( )'

การทำงานของ TSR นั้นส่วนใหญ่จะเป็นการทำงานโดยผ่าน 'window' ซึ่งก็คือส่วนหนึ่งของจอภาพที่ถูกแบ่งมาให้ TSR ใช้ ซึ่งก็จะทำให้สามารถเห็นจอภาพในส่วนที่ไม่ถูกนำ

มาใช้ได้ตามเดิม นอกจากนี้ TSR ยังถูกออกแบบมาเพื่อให้สามารถใช้งานร่วมกับโปรแกรมอื่น ๆ ได้ด้วย

วิธีการทำงานอีกอย่างหนึ่งของ TSR คือการเก็บหน่วยความจำของจอภาพในส่วนที่นำไปใช้งานเอาไว้ก่อน เพื่อที่จะสามารถนำมาแสดงเอาไว้อย่างเดิมหลังจากทำงานเสร็จแล้ว ซึ่งขั้นตอนเหล่านี้จำเป็นต้องใช้เวลา และเวลาที่ใช้นี้ก็จะเป็นสัดส่วนโดยตรงกับขนาดของจอภาพที่เก็บเอาไว้นั้น ฉะนั้นการออกแบบ TSR จึงควรที่จะให้มีการใช้เนื้อที่บนจอภาพน้อยที่สุดเพื่อให้สามารถทำงานได้เร็วขึ้น จากเหตุผลนี้ทำให้ TSR ต้องมีส่วนที่ทำหน้าที่จัดการกับวินโดว์เพื่อรับงานด้านการแสดงผลบนจอภาพ (screen I/O) ไปทำ อย่างไรก็ตามในบางกรณีอาจจะไม่สามารถใช้วินโดว์รูทีนต่าง ๆ ได้เพราะมีปัญหากับการจัดสรรหน่วยความจำ ซึ่งในเหตุการณ์แบบนี้อาจจำเป็นต้องใช้ I/O รูทีน

เพื่อจัดการกับวิดีโอแรมโดยตรง เพื่อให้สามารถคืนจอภาพได้เร็วที่สุดเท่าที่จะทำได้

สำหรับ TSR ที่เราจะได้พูดถึงกันต่อไปนั้น จะมีชุดคำสั่งของ windowing routines อย่างครบถ้วน รวมทั้งฟังก์ชันอื่น ๆ ที่ต้องใช้งานร่วมกัน ซึ่งจะเป็นประโยชน์กับการพัฒนา TSR ในโอกาสต่อไป

ขัดจังหวะดอสให้ปลอดภัย

โดยทั่วไปแล้ว การที่จะขัดจังหวะการทำงานของดอสได้อย่างปลอดภัย (ไม่ทำให้เกิดความเสียหายกับการทำงานของระบบที่กำลังทำอยู่) คือช่วงเวลาที่ดอสไม่ได้ทำอะไรที่สำคัญนัก เช่น เราสามารถที่จะขัดจังหวะดอสเมื่อกำลังแสดงเครื่องหมายพร้อมที่จะรับข้อมูลจากผู้ใช้ แต่ไม่ควรจะขัดจังหวะดอสในขณะที่กำลังติดต่อกับดิสก์หรือกำลังเขียนจอภาพ ซึ่งปัญหาอยู่ที่ว่าไมโครซอฟต์ไม่เคยให้รายละเอียด (ข้อมูล) เกี่ยวกับการทำงานของดอสว่าเมื่อไรที่ดอสอยู่ในสถานการณ์ที่เรียกได้ว่า 'ว่างงาน' (idle)

ในระยะเวลาที่ผ่านมาโปรแกรมเมอร์ทั้งหลายต่างก็พยายามค้นคว้าในเรื่องนี้และก็ได้พบว่ามีสิ่งน่ารู้อยู่ 2 อย่างที่ฟังก์ชันของดอสที่จะทำให้ TSR รู้ได้ว่าเมื่อไรที่ดอสปลอดภัยพอที่จะขัดจังหวะการทำงานของดอสได้ ซึ่งก็เป็นการค้นพบจากการทำงานของคำสั่ง 'PRINT' ของดอสนั่นเอง เพราะคุณสมบัติทั้งสองอย่างที่ว่านั้นใช้เพื่อสนับสนุนการทำงานของคำสั่งนี้ ซึ่งจะทำหน้าที่ทำงานอยู่หลังฉาก และคุณสมบัติเหล่านี้ไมโครซอฟต์ก็ไม่ได้แสดงไว้ให้รู้

DOS-Idle Interrupt : ในขณะที่ DOS กำลังรอรับข้อมูลจากผู้ใช้นั้น ดอสจะทำงานอยู่ในลูป (loop) เพื่อประมวลผลคำสั่งอินเตอร์รัพต์ 0×28 ซึ่งมักจะเรียกกันว่า 'DOS-idle interrupt' และอินเตอร์รัพต์นี้จะถูกเรียกใช้จากฟังก์ชันของดอส 12 ชุดแรก (ซึ่งจะทำหน้าที่เกี่ยวกับ คาแรกเตอร์ I/O) ฉะนั้นแต่ละครั้งที่อินเตอร์รัพต์ 0×28 ถูกประมวลผลก็จะหมายถึงการที่จะสามารถขัดจังหวะการทำงานของดอสได้อย่างปลอดภัย

ดังนั้นถ้าให้ TSR ขัดจังหวะการทำงานของดอส (ทำงานตามคำสั่งใน TSR) ในช่วงนี้ก็จะทำได้อย่างปลอดภัย อย่างไรก็ตามการตรวจสอบความปลอดภัยของระบบเพื่อเรียกใช้ TSR ด้วยการดูจากอินเตอร์รัพต์ 0×28 นี้ก็จะมีข้อยุ่งยากอยู่บ้าง เพราะดอสจะประมวลผลอินเตอร์รัพต์นี้เฉพาะ เมื่อเรียกใช้ฟังก์ชันใดฟังก์ชันหนึ่งในคาแรกเตอร์ I/O ฟังก์ชันเท่านั้น ฉะนั้นจึงจำเป็นต้องหาจุดอื่นที่จะทำให้รู้ได้ว่าสามารถขัดจังหวะดอสได้อย่างปลอดภัย ในกรณีที่ฟังก์ชันเหล่านี้ไม่ถูกเรียกใช้ด้วย เพราะการทำงานของโปรแกรมบางชุดจะเรียกใช้ BIOS โดยไม่ผ่านดอสเพื่อติดต่อกับ console I/O ซึ่งก็หมายความว่าจะไม่มีการประมวลผลอินเตอร์รัพต์ 0×28 เลยในขณะที่โปรแกรมเหล่านี้ทำงาน และผลที่ตามมาก็คือไม่สามารถจะใช้อินเตอร์รัพต์นี้ตรวจสอบสภาพความปลอดภัยในการเรียกใช้ TSR เพื่อให้ทำงานตามต้องการได้

DOS-Active Flag : คุณสมบัติอีกอย่างหนึ่งคือ เมื่อดอสกำลังทำงานจะกำหนดค่าของมันให้เป็น '1' และจะเปลี่ยนค่านี้ให้เป็น '0' เมื่อว่างงาน ซึ่งไบต์ที่มันใช้เพื่อกำหนดค่านี้คือ 'DOS-active flag' หรือบางครั้งก็จะเรียกว่า 'inDOS flag' โดยที่ค่าของ flag นี้จะเรียกดูได้ด้วยฟังก์ชัน 0×34 ของดอส ซึ่งจะทำหน้าที่คืนค่าเซกเมนต์ (segment) ของ 'DOS-active flag' นี้เอาไว้ในรีจีสเตอร์ 'ES' และค่าออฟเซต (offset) เอาไว้ในรีจีสเตอร์ 'BX' ฉะนั้นก่อนที่จะเรียกโปรแกรม TSR ก็จะต้องตรวจดูค่าของ flag นี้ถ้าพบว่ามีค่าเป็น '1' ก็ไม่ควรจะเรียกใช้ และจะต้องรอจนกว่าค่านี้จะเป็น '0'

อย่างไรก็ตาม เราไม่สามารถจะใช้ค่าของ flag นี้เพื่อตัดสินใจเรียกใช้ TSR เพียงอย่างเดียว เพราะในบางกรณีที่ flag นี้มีค่าเป็น '1' ก็ยังสามารถเรียกใช้ TSR ได้เช่นกัน (เมื่อดอสรอรับข้อมูลจากดอสพรอมต์ค่าของ flag นี้ก็จะถูกกำหนดให้เป็น '1' ด้วย) ซึ่งก็หมายความว่าเราจะต้องตรวจสอบอินเตอร์รัพต์ 0×28 ร่วมด้วย

ทางสู่มรรคผลของ TSR

อินเตอร์รัพต์อีกชุดหนึ่งที่เราจะต้องทำความรู้จัก เพื่อใช้ในการตรวจสอบความปลอดภัยที่จะขัดจังหวะการทำงานของดอสคือ 'timer interrupt' เพราะเครื่องคอมพิวเตอร์ไอบีเอ็มพีซีและเครื่องเลียนแบบทั้งหลายจะมีนาฬิกาของระบบที่จะทำการเรียกใช้อินเตอร์รัพต์ 9 ทุก ๆ 18.2 วินาที ซึ่งสามารถจะใช้อินเตอร์รัพต์นี้เพื่อตรวจสอบสถานะการทำงานของดอสว่ามีความปลอดภัยหรือไม่ เมื่อมีการเรียกใช้ (กด hotkey) TSR และเราจะได้เห็นวิธีการใช้อินเตอร์รัพต์ตัวนี้กับ TSR ในตัวอย่างของเราต่อไป

TSR ในโหมดกราฟิก

ถึงแม้จะไม่มีเหตุผลที่จะชี้ให้เห็นได้อย่างชัดเจนว่าไม่ควรจะเรียกใช้ (pop-up) TSR ในขณะที่เครื่องคอมพิวเตอร์กำลังทำงานอยู่ในโหมดกราฟิก แต่ก็จะเป็นการดีกว่า (และง่ายกว่าด้วย) ที่จะรอจนกว่าระบบจะกลับมาทำงานในเท็กซ์โหมด อย่างไรก็ตามเหตุผลที่พอจะนำมาสนับสนุนแนวทางการใช้ TSR ในแบบนี้ได้ก็คือ

* มีเท็กซ์โหมดอยู่เพียง 2-3 อย่าง และที่ใช้กันอยู่เป็นปกติก็คือ วิดีโอโหมด 2 (80-column black end white), โหมด 3 (80-column color text) และโหมด 7 (80-column mono-chrome) ส่วนในโหมดกราฟิกนั้นจะมีมากกว่า
* การทำงานใน 80-column text mode ทั้งหลายนั้นจะต้องการใช้หน่วยความจำเป็นจำนวน 4,000 ไบต์ เพื่อเก็บข้อมูลต่าง ๆ บนจอภาพ ไม่ว่าจะเป็นการใช้งานในโหมดไหนของ 80-column text mode

ซึ่งหมายความว่า เมื่อมีการเรียกใช้โปรแกรม TSR นั้น อย่างมากที่สุดจะต้องการหน่วยความจำ (buffer) เพียง 4,000 ไบต์เพื่อเก็บรายละเอียดของจอภาพทั้งหมด แต่เหตุการณ์แบบนี้จะไม่เป็นจริงกับการทำงานในโหมดกราฟิก เพราะมันต้องการใช้หน่วยความจำมากกว่า นอกจากนี้ในแต่ละโหมด (ของโหมดกราฟิก) ก็มีความต้องการใช้แรมไม่เท่ากัน ฉะนั้นการที่จะยอมให้ TSR ทำงานในโหมดกราฟิกนั้น จะหมายความถึงการที่เราจะต้องกันเนื้อที่หน่วยความจำไว้เป็นจำนวนเท่าที่จะต้องใช้กับโหมดกราฟิกที่ต้องการใช้เนื้อที่มากที่สุดเพื่อเก็บจอภาพ ซึ่งอาจจะมีขนาดใหญ่ถึง 64 กิโลไบต์ ไบต์ นอกเหนือจากเรื่องของขนาดหน่วยความจำแล้ว ยังมีเรื่องของเวลาที่จะต้องใช้ในการเก็บและคืนจอภาพซึ่งจะต้องใช้มากขึ้น โดยเฉพาะอย่างยิ่งกับการทำงานในโหมดกราฟิกที่มีความละเอียดสูงมาก ๆ ฉะนั้นการใช้งาน TSR ที่เหมาะสมที่สุดก็ควรจะอยู่ในเท็กซ์โหมดถึงแม้ว่าจะไม่ใช่ข้อจำกัดที่แท้จริงของ TSR ก็ตาม

การทำงานของ TSR กับวีดีโอแรม

เพื่อให้การทำงานของ TSR เป็นไปด้วยความเร็ว พอที่จะสร้างความพอใจให้กับผู้ใช้ ก็ควรที่จะให้โปรแกรมทำงานโดยไม่ต้องผ่านดอสและ BIOS วิดีโอฟังก์ชันด้วยการเขียนสิ่งที่ต้องการลงบนวิดีโอแรมโดยตรง ซึ่งวิธีการแบบนี้จะทำให้สามารถแสดงตัวอักษรบนจอภาพได้เร็วกว่ามาก

สำหรับวิธีการแสดงผลบนจอภาพนั้น ตัวอักษรต่าง ๆ ที่ถูกแสดงอยู่บนจอภาพจะถูกเก็บเอาไว้ในแรมที่จะถูกกันเอาไว้ให้กับ display adapters ในตำแหน่งหน่วยความจำของโมโนโครมก็จะอยู่ที่ 'B000:0000' ส่วน CGA/EGA/VGA นั้นตำแหน่งของวิดีโอแรมก็จะเริ่มที่ 'B800:0000' โดยที่ฟังก์ชันของอะแดปเตอร์เหล่านี้อาจจะมีความแตกต่างกันในบางโหมด แต่ก็จะเหมือนกันเมื่อทำงานในโหมด 2 หรือ 3

การแสดงผลของอักษรแต่ละตัวบนจอภาพนั้น จะต้องใช้หน่วยความจำในวิดีโอเมมโมรีเป็นจำนวน 2 ไบต์ โดยไบต์แรกจะใช้เก็บตัวอักษรที่จะแสดง ส่วนไบต์ที่สองนั้นจะใช้เก็บค่า screen attribute ซึ่งใช้แสดงลักษณะตัวอักษร เช่นแสดงสีหรือตัวกะพริบ สำหรับ color adepter นั้นไบต์ที่ใช้แสดงค่าแอตทริบิวต์จะเป็นไปตามที่แสดงไว้ในตารางที่ 1

ตารางที่ 1 แสดง text video attribute byte

Bit	Binary Value	Meaning whne Set
0	1	blue foreground
1	2	green foreground
2	4	red foreground
3	8	low intensity
4	16	blue background
5	32	green background
6	64	red background
7	128	blinking character

ซึ่งโดยปกติแล้ว การทำงานของ CGA, EGA และ VGA จะเป็นโหมด 3 และตัวอักษรจะถูกแสดงผลด้วยค่าแอตทริบิวต์เป็น 7 (1+2+4) ซึ่งจะเป็นการกำหนดให้บิตที่ใช้แสดงสีของตัวอักษรทั้ง 3 สี คือ น้ำเงิน, เขียว และแดง เป็น 'on' (ถูกกำหนดให้มีค่าเป็น '1') ซึ่งจะมีผลให้ตัวอักษรที่แสดงออกมาเป็นสีขาว หรือถ้าต้องการให้แสดงสลับสีก็ให้กำหนดค่าของบิตที่ใช้แสดงสีพื้นให้เป็น 'off' (กำหนดให้เป็น '0') และให้บิตที่แสดงสีพื้นทั้ง 3 ตัวเป็น 'on' ซึ่งก็จะทำให้แอตทริบิวต์ไบต์มีค่าเป็น 70 H

ในกรณีของโมโนโครมอะแดปเตอร์จะแสดงค่าแอตทริบิวต์ได้เฉพาะ 'blinking' (กะพริบ) และ 'intensity' (สว่างกว่าปกติ) อย่างไรก็ตามก็ยังสามารถที่จะใช้ค่า 7 สำหรับการแสดงผลตามปกติ และ 70H สำหรับการแสดงในแบบสลับสี รวมถึงค่า 1 ที่ใช้แสดงตัวอักษรแบบขีดเส้นใต้

จำนวนหน่วยความจำที่อะแดปเตอร์แต่ละชนิด (ยกเว้นโมโนโครมอะแดปเตอร์) ต้องการใช้นั้นจะมีขนาดเป็น 4 เท่าของหน่วยความจำที่ใช้สำหรับการทำงานใน Text 80-column mode ซึ่งเหตุผลคือ

* ความจำส่วนที่เกินมานั้นจะใช้สำหรับการทำงานในโหมดกราฟิก
* เพื่อใช้งานแบบหลายจอภาพในแรม ซึ่งสามารถจะเรียกใช้แต่ละจอภาพได้ด้วยการเรียกใช้ (switch) ในส่วนที่ต้องการ โดยแต่ละส่วน (จอภาพ) นั้นจะเรียกว่า 'video page'

การใช้งานวิดีโอแรมของอะแดปเตอร์ต่าง ๆ นั้น ถ้าไม่มีการกำหนดให้เป็นอย่างอื่น ก็จะเรียกใช้ page 0 เสมอ และโดยปกติแล้วโปรแกรมต่าง ๆ ก็จะใช้ page นี้เป็นส่วนใช้งานเพื่อการแสดงผลบนจอภาพเช่นกัน ซึ่ง TSR ที่เราจะพูดถึงกันต่อไปก็จะใช้ page 0 ด้วย แต่ถ้าคุณต้องการจะเปลี่ยนไปใช้งานอื่นก็ย่อมได้

ถ้าต้องการจะรู้ตำแหน่งของวิดีโอแรมจำเป็นจะต้องรู้ว่ากำลังใช้วิดีโอแรมอะไรอยู่ เพราะโมโนโครมอะแดปเตอร์และอะแดปเตอร์อื่น ๆ จะมีตำแหน่งเริ่มต้นไม่เหมือนกัน แต่มีวิธีตรวจสอบได้จาก ROM-BIOS อินเตอร์รัพต์ 16 โดยเรียกใช้ฟังก์ชัน 16 ซึ่งจะคืนค่าของวิดีโอโหมดที่ใช้งานอยู่ในขณะนั้นมาให้ และถ้าเป็นการใช้งานใน text 80-column mode แล้ว ก็จะคืนค่า 2, 3 หรือ 7 มาให้ ซึ่งโหมด 2 และ 3 สามารถใช้ได้เฉพาะกับ CGA, EGA และ VGA เท่านั้น โดยไม่สามารถจะใช้โหมด 7 ได้ เพราะเป็นโหมดที่ใช้งานสำหรับโมโนโครมอะแดปเตอร์ ฉะนั้นถ้าค่าที่คืนมาเป็น 7 ก็แสดงว่าเป็นโมโนโครมอะแดปเตอร์ แต่ถ้าเป็น 2 หรือ 3 ก็อาจจะเป็นได้ทั้ง CGA, EGA หรือ VGA และเมื่อรู้ว่าเป็นวิดีโอโหมดแบบไหนแล้ว ก็สามารถที่จะกำหนดค่าของตัวแปรให้เท่ากับตำแหน่งเริ่มต้นที่เหมาะสมได้

เมื่อมีการกำหนดเริ่มต้นของวิดีโอแรมให้กับตัวชี้เรียบร้อยแล้ว ก็สามารถที่จะใช้พื้นที่นั้นอ่านหรือเขียนข้อมูลได้ แต่จะต้องระลึกอยู่เสมอว่าวิดีโอเมมโมรีนั้นต้องการใช้เนื้อที่ 2 ไบต์สำหรับอักษรแต่ละตัว ซึ่งก็หมายความว่าจะใช้เนื้อที่จำนวน 160 ไบต์ สำหรับข้อมูลแต่ละแถวในจอภาพ (80 ตัวอักษรต่อบรรทัด) ฉะนั้นถ้าต้องการหาตำแหน่งของตัวอักษรที่ต้องการ ก็สามารถจะใช้สูตรดังต่อไปนี้ได้

```
address = address OF adapter
+ X * 160 + Y * 2
```

เมื่อแอดเดรสคือตำแหน่งที่ต้องการรู้ และแอดเดรสของอะแดปเตอร์คือตำแหน่งเริ่มต้นของวิดีโอแรมส่วน X คือจำนวนแถว (row) และ Y คือจำนวนคอลัมน์ (column)

อาวุธของ TSR

คอมพายเลอร์ เทอร์โบซีได้เพิ่มฟังก์ชันของตัวเองอีกหลายชุด สำหรับให้ความสะดวกกับโปรแกรมเมอร์ที่จะเขียนโปรแกรม TSR ด้วยภาษาซี และการเขียนโปรแกรม TSR ที่เราจะได้พูดถึงกันต่อไปนั้น จะใช้ฟังก์ชันพิเศษของเทอร์โบซีที่สร้างขึ้นไว้เพื่อสนับสนุนการเขียนโปรแกรมในลักษณะนี้ (ซึ่งคอมพายเลอร์หลาย ๆ ตัวก็มีฟังก์ชันที่ทำงานในลักษณะที่ว่านี้เช่นกัน) ด้วย ซึ่งก็ได้แก่ 'int86( )', 'int86x( )', 'setvect( )' และ 'freemem( )' โดยที่แต่ละฟังก์ชันก็จะทำหน้าที่ต่าง ๆ กันดังนี้

int86( ): ใช้ทำหน้าที่สร้างอินเตอร์รัพต์ให้กับโปรแกรม ซึ่งจะมีรูปแบบการใช้งานดังรูปที่ 1

```
int int 86 (int num, /* the interrupt number */
        union REGS * inrege /* the input register values */
        union REGS * outtrege /* the output register values */
```

รูปที่ 1 แสดงรูปแบบการใช้งานของ int 86 ( )

ซึ่งอินเตอร์รัพต์ที่จะถูกสร้างขึ้นมานั้น จะกำหนดด้วยตัวแปร 'num' และสามารถจะกำหนดค่าของรีจีสเตอร์ต่างๆ ในระหว่างการเรียกใช้อินเตอร์รัพต์ได้ด้วยตัวแปร '*inregs' ส่วนการคืนค่าที่ถูกกำหนดด้วย TSR (โดยเก็บค่านี้เอาไว้ในรีจีสเตอร์) ที่ทำหน้าที่ควบคุมการอินเตอร์รัพต์นั้นจะใช้ตัวแปร '*outtregs' ซึ่งค่าที่ฟังก์ชัน 'int86( )' จะส่งคืนมาให้นั้นคือค่าที่อยู่ในรีจีสเตอร์ 'AX' ส่วน 'REGS' นั้นคือชนิดของตัวแปร กำหนดเอาไว้ในส่วนหัว (header) ของไฟล์ 'DOS.H' ซึ่งถูกกำหนดเอาไว้ให้ใช้ในเทอร์โบซี แต่ในคอมพายเลอร์ตัวอื่น ๆ เช่น ไมโครซอฟต์ C ก็มีการกำหนดในลักษณะที่คล้าย ๆ กันนี้ด้วย ดังรูปที่ 2

```
/*
Copyright (C) Borland International Inc, 1987
All Rights Reserved.
*/
struct WORDREGS
        {
        unsigned int ax, bx, cx, dx, si, di cflag;
        }
union REGS {
        struct WORDREGS X;
        struct BYTEREGS h;
        };
```

รูปที่ 2 แสดงรูปแบบของโปรแกรมในไมโครซอฟต์ซี

ซึ่งจะเห็นได้ว่า 'REGS' เป็นส่วนประกอบของตัวแปร 2 ชนิด ซึ่งการใช้งาน 'WORDREGS' นั้น จะทำให้สามารถเรียกใช้รีจีสเตอร์ของซีพียูในขนาด 16 บิต ส่วนการใช้ 'BYTEREGS' นั้นจะเป็นการเรียกใช้รีจีสเตอร์เพียง 8 บิต ตัวอย่างเช่นถ้าต้องการเรียกใช้อินเตอร์รัพต์ 16, ฟังก์ชัน 5 ก็จะสามารถทำได้ด้วยการเขียนคำสั่งดังนี้

```
union REGS in, out;
in.h.eh = 5;
int86 (16, &in, &out);
```

int86x( ): การทำงานของฟังก์ชันนี้ก็จะคล้ายกับฟังก์ชัน 'int86( )' ที่ได้พูดถึงไปแล้วคือจะทำการดึงเอาค่าของเซกเมนต์รีจีสเตอร์เข้าไปเก็บไว้ในตัวแปรที่กำหนดให้เป็นแบบ (type) 'SREGS' ซึ่งเป็นอาร์กิวเมนต์ตัวที่ 3

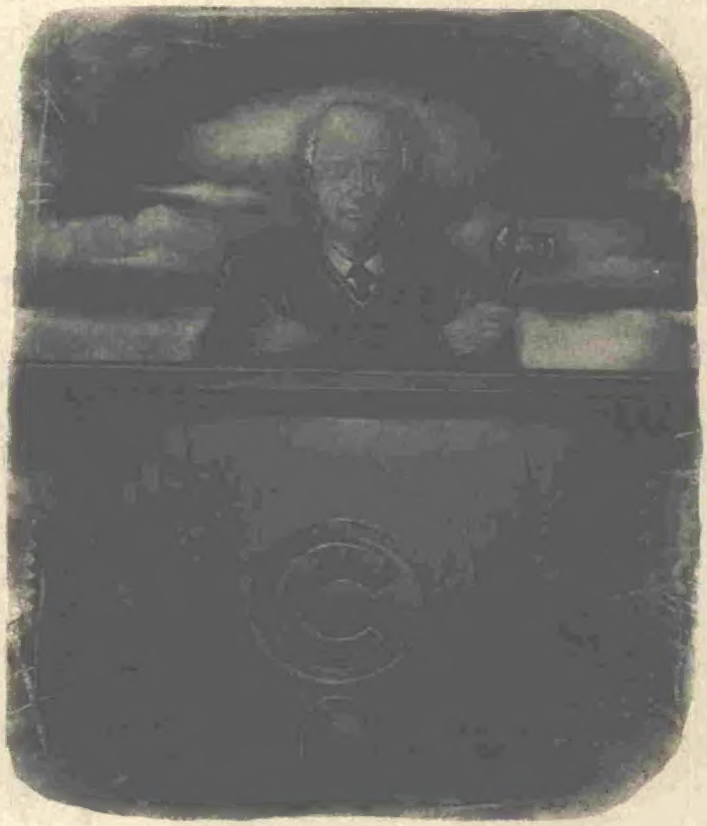
getvect ( ) และ setvect ( ): เป็นฟังก์ชันที่จะถูกใช้เพื่ออ่านค่าและกำหนดค่าตามลำดับ ซึ่งค่าที่มันจะใช้คือตำแหน่งที่อยู่ในตารางอินเตอร์รัพต์ โดยจะมีรูปแบบของฟังก์ชันดังนี้

void interrupt getvect (int int-num);

void setvect (int int-num, void interrupt (*ISR) ( ));

ซึ่งในที่นี้ 'ISR' คือตัวชี้ที่จะชี้ไปที่ interrupt service routine (ISR) ชุดใหม่ที่ต้องการ ซึ่งถูกเก็บตำแหน่งในตัวแปร 'int_num' ของตารางเวกเตอร์เวกเตอร์

ตัวอย่างเช่นถ้าต้องการคืนค่า



ตำแหน่งของ ISR ที่อินเตอร์รัพต์ 5 ใช้อยู่ ก็ให้เขียนคำสั่งตามนี้

```
addr = getvect (5):
```

และจะกำหนดค่าอินเตอร์รัพต์ 5 ของ ISR ให้กับตำแหน่งของฟังก์ชันชื่อ 'newints ( )' ก็จะทำได้ดังนี้

```
setvect (5, newint5):
```

freemem ( ): สำหรับฟังก์ชันนี้จะทำหน้าที่คืนหน่วยความจำที่ดอสเคยจัดสรรให้ไว้ก่อนแล้ว ซึ่งมันจะเรียกใช้ฟังก์ชัน 0x49 ของดอส โดยที่ค่าอาร์กิวเมนต์ของมันคือตำแหน่งของหน่วยความจำที่ต้องการจะคืนให้กับดอส

พักกันก่อน

สำหรับเรื่องราวของการเขียนโปรแกรมแบบ TSR (Terminate and Stay Resident) ด้วยภาษาซีของเราในคราวนี้ก็คงจะต้องหยุดเอาไว้แค่นี้ก่อน แต่ก่อนจะพักก็ลองมาดูกันก่อนว่าเราได้พูดกันถึงเรื่องอะไรไว้บ้างแล้ว

จากจุดเริ่มต้นเราได้ทำความรู้จักกับโปรแกรมประเภท TSR และการเขียนโปรแกรมประเภทนี้ให้ทำงานกับดอส (PC-DOS) ซึ่งเป็นเรื่องราวของความขัดแย้งกันระหว่างการทำงานของ TSR กับดอสรวมถึงวิธีการที่จะใช้เพื่อแก้ไขปัญหาความขัดแย้งนั้นและการหาวิธีให้ TSR สามารถทำงานได้โดยไม่ทำความเสียหายให้กับงานที่กำลังทำอยู่ก่อนการเรียกใช้ TSR

นอกจากนี้ได้พูดถึงขีดความสามารถของโปรแกรม TSR รวมถึงวิธีการตรวจสอบสถานะการทำงานของดอส การเรียกใช้อินเตอร์รัพต์ที่จำเป็นและ interrupt type modifier ของเทอร์โบซี แล้วยังมีเรื่องของ PSP (Program Segment Prefix) ที่ TSR จะต้องเข้าไปเกี่ยวข้องโดยใกล้ชิดด้วย

ต่อจากนั้นก็เป็นเรื่องราวของการหาวิธีที่จะขัดจังหวะการทำงานของดอสให้มีความปลอดภัยมากที่สุด แล้วยังมีเรื่องของความเกี่ยวพันกันระหว่างการทำงานของ TSR กับกราฟิกโหมด รวมถึงการทำความรู้จักกับวิดีโอแรม ซึ่งมีความสัมพันธ์โดยใกล้ชิดกับโปรแกรม TSR และปิดท้ายด้วยเรื่องของฟังก์ชันบางชุดของเทอร์โบซีที่มีส่วนช่วยให้การเขียนโปรแกรม TSR ทำได้สะดวกขึ้น

ในบทความชุดต่อไป (ตอนที่ 2) จะได้มาดูกันถึงวิธีการเขียนโปรแกรม TSR ซึ่งก็จะเป็นเรื่องราวเกี่ยวกับการกำหนดค่าเริ่มต้นให้กับ TSR รวมทั้งการทำความรู้จักกับฟังก์ชัน 'ter-key-stroke( )' และ keystroke character buffer แล้วยังมีเรื่องของวิธีการสร้างอินเตอร์รัพต์, TSR และฟังก์ชันของ TSR ที่จะได้สร้างขึ้นมาเป็นตัวอย่างหรือแนวทางที่จะใช้เป็นหลักในการเขียนโปรแกรม TSR เพราะในบทความตอนแรกนี้ก็ได้ปูพื้นฐานเกี่ยวกับ TSR และฟังก์ชันต่าง ๆ ของเทอร์โบซีเอาไว้พอควรแล้ว ฉะนั้นในคราวหน้าเราก็จะมาลองสร้าง TSR กันดู ขอให้ติดตามกันต่อไปนะครับ